Quantum Inf Process (2017) 16:245
DOI 10.1007/s11128-017-1692-x

A quantum-implementable neural network model

Jialin Chen[1,2] · **Lingli Wang**[1] ·
Edoardo Charbon[3,4]

Received: 7 February 2017 / Accepted: 9 August 2017 / Published online: 24 August 2017

Abstract A quantum-implementable neural network, namely quantum probability neural network (QPNN) model, is proposed in this paper. QPNN can use quantum parallelism to trace all possible network states to improve the result. Due to its unique quantum nature, this model is robust to several quantum noises under certain conditions, which can be efficiently implemented by the qubus quantum computer. Another advantage is that QPNN can be used as memory to retrieve the most relevant data and even to generate new data. The MATLAB experimental results of Iris data classification and MNIST handwriting recognition show that much less neuron resources are required in QPNN to obtain a good result than the classical feedforward neural network. The proposed QPNN model indicates that quantum effects are useful for real-life classification tasks.

Keywords Feedforward neural network · Quantum probability neural network · Quantum neuron · Iris and MNIST experiments

1 Introduction

Quantum computation has been one of the fastest growing areas in physics during last decades. The quantum computer was originally proposed for quantum mechanics

✉ Lingli Wang
llwang@fudan.edu.cn

1 State Key Laboratory of ASIC and System, Fudan University, Shanghai, People's Republic of China

2 School of Computer Science, Fudan University, Shanghai, People's Republic of China

3 QuTech, Delft, The Netherlands

4 Advanced Quantum Architecture Lab (AQUA), EPFL, Lausanne, Switzerland

research by Richard Feynman [1]. However, many quantum effects such as entanglement and superposition indicate that such computers are also exponentially or geometrically faster than their classical counterparts in some applications, such as integer factorization [2], database search [3], and global binary optimization [4].

However, due to the following reasons, the number of applications that quantum computers can solve efficiently is still limited. Firstly, the laws of quantum mechanics restrict the access to information stored in quantum systems. Although n quantum bits, or qubits, can store a superposition state with 2^n different classical states, only one of these states can be observed with a certain probability. Secondly, algorithm design for quantum computers is hard as the human thinking is rooted in the classical world, where the impressive power of quantum parallelism cannot be easily understood and fully utilized for most applications. In fact, in order to design algorithms for applications, we often potentially use the idea of functions in which the output depends on current input states rather than the superposition of all possible input states. Thirdly, a useful quantum information processing system is hard to build, due to the requirement of certain error correction mechanisms, low temperatures, the special materials, in most cases. Moreover, many studies of quantum algorithms focus on the running speed compared to their classical partners, such as exponential or quadratic speedups. In our view, some problems in certain fields like machine learning may be solved efficiently on quantum computers in aspects other than the speed, such as pattern recognition, knowledge discovery [5], and memory capacity [6].

Machine learning can process large amounts of information for tasks that come natural to the human brain, such as image and speech recognition, pattern identification, or strategy optimization [7]. As one of the most famous machine learning algorithms, artificial neural network (ANN) is a model of computation inspired by the structure of neural networks in brains. Like the simplified brain's model, ANN is an n-dimensional graph where the nodes are called neurons and their connections are weighted by parameters. From the mid-twentieth century, it has become an effective learning paradigm and has recently been shown to achieve cutting-edge performance on several learning tasks [5]. However, in the era of big data, the pressure to find innovative approaches to machine learning is rising, whereas combining machine learning and quantum computation is an open and fascinating topic of research.

Several recent academic contributions have explored the idea of using the advantages of quantum computation to improve machine learning algorithms. A simple and successful way is to run subroutines of classical machine learning algorithms such as K-means, K-nearest neighbors, and support vector machine (SVM) on a quantum computer to gain significant speedup [8–10]. However, as the nonlinear and dissipative dynamics of classical neurons, most kinds of ANNs such as feedforward neural networks (FFNNs) cannot be simulated by linear and unitary dynamics of quantum computation directly. As a result, in an early stage, most of quantum neural network (QNN) proposals [11,12] consider a special kind of ANNs, called Hopfield networks, which are based on the associative memory [13] rather than nonlinear activation function. Associative memory is the ability to retrieve the network state out of n stored network states which is closest to the input pattern in terms of Hamming distance. In fact, few real-life applications depend on the Hamming distance, thus limiting the usage of the Hopfield QNN. Later, many proposals [14–18] attempt to find a quantum equivalent for the perceptron or sigmoid neurons from which neural networks are constructed. Compared to classical ANNs, QNNs have shown advantages in some tasks such as the Iris data classification [17] and nonlinear function fitting [18]. However, most of them introduce nonlinear operators in their structures. Although nonlinear quantum mechanics has been studied for years [19], the physical implementation of nonlinear quantum computing is still controversial. Although reference [14] shows that the time evolution of interacting quantum dots can simulate the nonlinear dynamics of NN, it is still hard to build a complete NN structure [16]. Moreover, it is a hardware-dependent model, rather than a universal model. Therefore, the above proposals are out of our scope in this paper. Recently, reference [20] has proposed a quantum perceptron over a field based on quantum gates. However, this perceptron is without an activation function and no experimental result is provided to verify its performance in applications. Only reference [21] has proposed a direct implementation of the perceptron function using the quantum circuit model based on the quantum phase estimation algorithm. Obviously, the direct quantization of classical neuron has few advantages of the quantum computation, but it can be used as a building block of other QNNs in the future. In fact, QNN research is still in its infancy. Even a precise definition of what is a quantum neural network that integrates neural computation and quantum computation is a nontrivial open problem [20].

In this paper, we follow the unitary principles of quantum computation so that nonunitary operators here are measurements only. To realize the quantum-implementable neural network, particularly for FFNN, QPNN is proposed. It is composed of a new type of quantum neurons, or qurons, and their connections. QPNN can utilize quantum parallelism to trace all possible network states and even create many networks with different parameters to improve the result. Therefore, they have unique advantages over classical FFNNs. To verify this idea, we apply QPNN to two real-life classification applications, i.e., Iris data classification [22] and MNIST handwritten digit database recognition [23]. Here, in both experiments, MATLAB simulation results show that much less quron resources are required for QPNNs to achieve a good classification accuracy than for classical FFNNs. As far as we know, no experiment like MNIST on QNNs with a dataset of the similar size has been reported to compare with. In addition to the resources saving, QPNN can also be used as memory to retrieve the most relevant data and even to generate new data. In view of the physical implementation, QPNN is robust to certain quantum noise sources, such as phase flip channel and bit-flip channel [1], which can be efficiently implemented by universal quantum computers.

The paper is organized as follows: Sect. 2 describes the conventions used throughout this paper and a brief review of ANNs. Section 3 details the structure of QPNN and its simulation flow. In Sect. 4, we show how this structure can be implemented very efficiently by the qubus quantum computer [24,25]. The learning algorithm of QPNN is presented in Sect. 5. Section 6 discusses the application of QPNN to the Iris data classification and MNIST handwritten digit recognition. Conclusion and challenges are presented in Sect. 7.

2 Preliminary

2.1 Conventions

In the context of machine learning, x_i^n represents the ith component value of the nth training sample; ω_j^i represents the weight from input neuron j to output neuron i; f' indicates the derivative or partial derivative of function f with respect to certain variables.

In the context of quantum computation, the Dirac notation is used to express variables as they are prepared in quantum states. For example, $|x^n\rangle$ indicates the nth training sample which is prepared in the quantum state with the value of x^n. Moreover, y_k^n is used to denote the output conditional probability $p\left(|y^n\rangle = |k\rangle \, |x^n\right)$, or $p\left(|y^n\rangle = |k\rangle\right)$ for short, where $k \in \mathbb{N}$.

In neural networks, especially their quantum version, a large number of tensor calculations are required. Hence, a modified version*of Einstein notation* [26] is used to achieve notational brevity. That is, if an index variable appears more than once in the subscript position or the superscript position in a single term, it implies summation of that term over all the values of the index. For example,

$$y = \sum_{i=1}^{3} c_i x^i d_i = c_1 x^1 d_1 + c_2 x^2 d_2 + c_3 x^3 d_3 \tag{1}$$

is simplified by the convention

$$y = c_i x^i d_i \tag{2}$$

The upper indices are not exponents, but are indices of coordinates or dual vectors. Moreover, we use $x = (x_1 \cdots x_m)_2$ with $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, m$ to denote the binary expansion of the number x.

2.2 Artificial neural networks review

Inspired by the behavior of biological neurons, an artificial neural network is a set of interconnected neurons and their connections whose values are obtained by certain learning rules [27]. The activation mechanism of a neuron j based on input neurons $x_1, \ldots, x_n$ forms the core of the ANN. There are several ANN activation functions. For example, the *perceptron* function is

$$y_j = \begin{cases} 1, & \text{if} \quad x \cdot \omega^j \geq 0 \\ -1, & \text{else} \end{cases} \tag{3}$$

where $x = (x_1, \ldots, x_n)$ and ω^j represents weights from the previous layer to the neuron j. The above function means that the input signals are simply added up by weights and compared with the threshold 0. If the result exceeds the threshold, j is activated; otherwise, j rests. These kinds of neurons are also called McCulloch–Pitts neurons [5]. A neural network of McCulloch–Pitts neurons in which the connectivity obeys the following equation

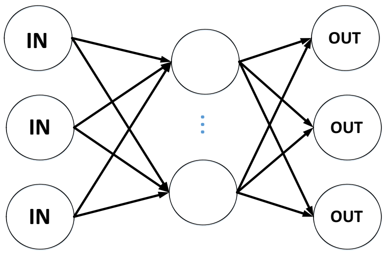


Fig. 1 Feedforward neural network

$$\omega_j^i = \omega_i^j, \omega_i^i = 0 \tag{4}$$

is called *Hopfield neural network (HNN)* [7]. The Hopfield model shows the powerful feature of the associative memory. Associative memory is the ability to retrieve the network state out of P stored network states which is closest to the input pattern in terms of Hamming distance. As the associative memory can be realized by Grover algorithm, most quantum versions of neural networks are based on HNN.

For pattern classification, the most widely used activation function is the sigmoid function

$$y_j = \mathrm{sgm}\left(x \cdot \omega^j\right) \tag{5}$$

where $\mathrm{sgm}\,(\alpha) = \left(1 + e^{-\alpha}\right)^{-1}$. Instead of the binary values of a McCulloch–Pitts neuron, this kind of neurons can take values out of a continuous range (0, 1).

The sigmoid activation function is often used in FFNN to update neurons. In FFNN, neurons are arranged in layers, and each layer feeds its values into the next layer. Each layer successively updates its neurons from the previous layer's outputs, and the encoding of the classification can be read out in the last layer, as shown in Fig. 1.

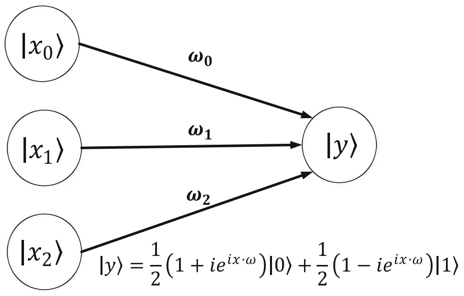
What has attracted the most interest in neural networks is the ability of learning. Given a specific task to solve and a hypothesis class H, learning means using a set of training samples to find $h \in H$ which solves the task in some optimal sense, i.e., no solution has a cost less than the cost of the optimal solution. Therefore, the loss function L should be defined as a measure of distance between the output solution and the optimal solution according to the defined cost. Then, by minimizing L, the learning algorithm searches through the solution space to find a function that has the smallest possible cost.

For many applications, given a proper loss function of outputs such as the squared error for regression or the cross-entropy for classification, through the back-propagation algorithm [5] we can always find a set of weight parameters, which make FFNNs classify input patters extremely well.

3 Quantum probability neural network

Quantum mechanics is a linear theory, which means that maps of one state onto another are executed by linear operators. Furthermore, in order to ensure probability

Fig. 2 Diagram of a quron


$|x_0\rangle$
$|x_1\rangle$
$|x_2\rangle$
$|y\rangle$
ω_0
ω_1
ω_2
$|y\rangle = \frac{1}{2}(1 + ie^{ix\cdot\omega})|0\rangle + \frac{1}{2}(1 - ie^{ix\cdot\omega})|1\rangle$

conservation, the operator U has to be unitary, $U^{\dagger}U = I$. In fact, the fundamental challenge of quantum neural network is the integration of the nonlinear, dissipative dynamics of classical neurons and linear, unitary quantum theory. As mentioned in Sect. 2, ANNs based on the sigmoid function are widely used in practice. However, this kind of activation function cannot be realized by any linear quantum operators or gates.

To partially solve this problem, we propose a new kind of neurons, or quron, based on quantum unitary transformation, described as follows.

As shown in Fig. 2, when each input quron is in a computation basis state, the quron y is in a superposition state of $|0\rangle$ and $|1\rangle$, or "rest" and "active" as given by

$$|y\rangle = \frac{1}{2}\left(1 + i\mathrm{e}^{i(x\cdot\omega+b)}\right)|0\rangle + \frac{1}{2}\left(1 - i\mathrm{e}^{i(x\cdot\omega+b)}\right)|1\rangle \tag{6}$$

where $x = (x_0, x_1, x_2) \in \mathbb{R}^3$ and $\omega = (\omega_0, \omega_1, \omega_2) \in \mathbb{R}^3$. So, probabilities of "0" and "1" are $\left|\frac{1}{2}\left(1 + i\mathrm{e}^{i(x\cdot\omega+b)}\right)\right|^2 = \frac{1}{2} - \frac{1}{2}\sin(x\cdot\omega + b)$ and $\left|\frac{1}{2}\left(1 - i\mathrm{e}^{i(x\cdot\omega+b)}\right)\right|^2 = \frac{1}{2} + \frac{1}{2}\sin(x\cdot\omega + b)$, respectively. It may be more convenient to incorporate b,or *bias*, into ω as an extra coordinate and add an extra coordinate with a value of 1 to x. Namely, let $\omega' = (b, \omega_1, \ldots, \omega_d)$ and $x' = (1, x_1, \ldots, x_d)$, then $x\cdot\omega + b = x'.\omega'$. Due to the periodicity of trigonometric functions, we can restrict $x'.\omega' \in [-\pi, \pi]$. Therefore, if input x' satisfies $x'.\omega' \geq 0$, then the output quron will be activated with a probability larger than 0.5 and rest otherwise. In this sense, qurons are similar to classical neurons based on the sigmoid function.

Obviously, Eq. (6) can be extended to a general situation where each input quron $|x\rangle$ is in a superposition state. Consider, for example, an input layer with n qurons where each of them is prepared in a superposition state of $|0\rangle$ and $|1\rangle$, i.e., each quron is a single qubit; then, the tensor state can be written as $\sum_{i=0}^{2^n-1} c_i\,|i\rangle = \left|x^0\right\rangle \otimes \cdots \otimes |x^n\rangle$ where $|i\rangle$ is the ith computational basis for a 2^n-dimensional Hilbert space. By Eq. (6), the whole system state in Fig. 2 is

$$|\psi\rangle = \sum_i c_i\,|i\rangle \otimes \left(f_1\left(i_B\cdot\omega\right)|1\rangle + f_0\left(i_B\cdot\omega\right)|0\rangle\right) \tag{7}$$

where $f_0(x) = \frac{1}{2}\left(1 + i\mathrm{e}^{ix}\right)$, $f_1(x) = \frac{1}{2}\left(1 - i\mathrm{e}^{ix}\right)$ and i_B is a binary expansion vector of the number i. For example, if $i = 2$, then $i_B = (1, 0)$.

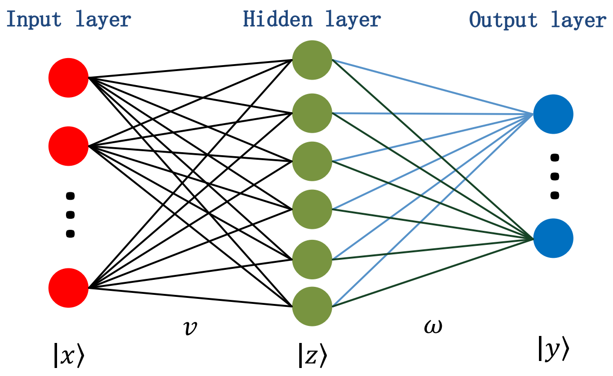


Fig. 3 A two-layer structure of QPNN

Based on the above dynamics of qurons, any topological structure of networks can be constructed. But before that, an important question is the representation of sample data by qubits. In QPNN, sample data $x \in \mathbb{R}$ are stored in a quron which comprises of d qubits. This d-qubit state $|x_1, \ldots, x_d\rangle$ is directly translated by its binary expansion $(x_1 \ldots x_d)_2$ with $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, d$. So, if $0 < x < 1$, the numerical accuracy of x is no more than 2^{-d}.

For simplicity, here we restrict our attention to a two-layer fully connected network structure for the classification task where the hidden layer and the output layer are comprised of m qurons and p qurons, respectively, as shown in Fig. 3. Moreover, each of the qurons at hidden and output layer is a qubit. For the purpose of simulating the dynamics of QPNN, we need to define the*layer transition matrix*, or *LTM*, first.

Definition Let k_n represent the number of qurons in layer n, and the *layer transition matrix* β_n is a $2^{k_{n-1}} \times 2^{k_n}$matrix where $\beta_i^j = \langle i|\beta|j\rangle$ denotes the probability of layer n in state $|i\rangle$ when layer $(n-1)$ is in state $|j\rangle$. For $n = 1$, β_1 is a row vector of the probability distribution of the input states.

Using LTM, the measurement result of $|y\rangle$ in Eq. (7) can be written simply as

$$p\,(y = 1) = (\beta_1)_i\,(\beta_2)_1^i \tag{8}$$

where $(\beta_1)_i = |c_i|^2$ and $(\beta_2)_1^i = |f_1\,(i_B \cdot \omega)|^2$ represent the LTM of layer 1 and layer 2, respectively. It should be mentioned again that all equations in the paper use the *summation convention* described in Sect. 2.

Now we proceed with the simulation in a layer-by-layer manner. Suppose the nth training sample is a d-dimension vector x^n, then the quron j at input layer is prepared in the basis state $\left|x_j^n\right\rangle$ for $j = 1, \ldots, d$. According to Eq. (6), the quron i at hidden layer becomes

$$\left|z_i^{\mathrm{n}}\right\rangle = f_0\left(x^n \cdot v^i\right)|0\rangle + f_1\left(x^n \cdot v^i\right)|1\rangle \tag{9}$$

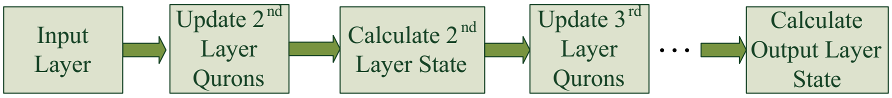


Fig. 4 Classical simulation flow of a general QPNN's forward pass

where the vector v^i represents weights of the ith quron at hidden layer. Therefore, for a hidden layer which is comprised of m qurons, its layer state can be represented as $\sum_{i=0}^{2^m-1} c_i^n \left|i\right\rangle = \left|z_1^n\right\rangle \otimes \cdots \otimes \left|z_m^n\right\rangle$. Next, if ω^j represents weights of the quron j at output layer, then the state of it becomes

$$\left|y_j^n\right\rangle = \sum_i c_i^n f_1\left(i_B \cdot \omega^j\right)\left|1\right\rangle + c_i^n f_0\left(i_B \cdot \omega^j\right)\left|0\right\rangle \tag{10}$$

For a multi-class classification application, several output qurons are needed to represent the probability distribution of class labels by their tension product state.

Finally, we measure the output layer with respect to the computation basis to get the classification result. For the nth training sample x^n, the conditional probability of result $k = \left(k_1 \cdots k_p\right)_2$ is

$$p\left(y^n = k|x^n\right) = \sum_{i=0}^{2^m-1} \left(\beta_2\right)_i^n \left(\beta_3\right)_{k=\left(k_1\cdots k_p\right)_2}^i = \sum_{i=0}^{2^m-1} \left| c_i^n \prod_{j=1}^{p} f_{\mathrm{k}_j}\left(i_B \cdot \omega^j\right)\right|^2 \tag{11}$$

where β_2 and β_3 are LTM of layer 2 (hidden layer) and layer 3 (output layer), respectively. Figure 4 shows the classical simulation flow of a general QPNN's forward pass.

In Eq. (11), the summation index i is explicitly written because it conveys the basic idea of QPNN: The output is the expectation value of all 2^m possible states of the hidden layer. Therefore, QPNN cannot be simulated efficiently by classical computers. In contrast, the kth output neuron of a two-layer sigmoid-based FFNN is

$$y_k^n = \mathrm{sgm}\left(\mathrm{sgm}\left(x^n \cdot v^j\right)\omega_j^k\right) \tag{12}$$

Therefore, in a sense, the classical FFNN only considers the situation in which all the hidden layer neurons are activated, while QPNN traces all possible situations simultaneously and computes their expectation value.

One of the most interesting points of QPNN is that we can use quantum parallelism to combine several basic networks at the same time. To achieve this purpose, only n qubits are needed in a control layer to perform the following quantum multiplexer as shown in Fig. 5

$$\begin{pmatrix} U_1 & & \\ & \ddots & \\ & & U_{2^n} \end{pmatrix} \tag{13}$$

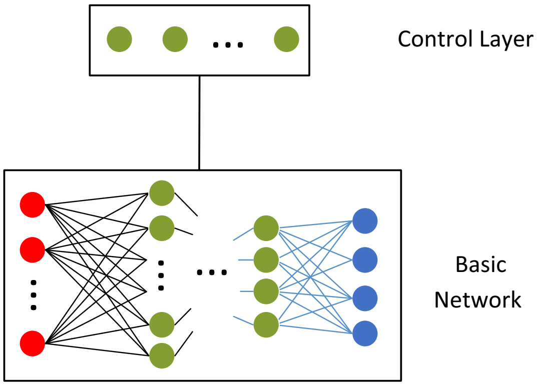


Fig. 5 QPNN structure with a control layer

where U_i represents the dynamics of the ith basic network. Moreover, 2^n different quantum gates $\left\{P^{(i)} \middle| \quad i=1,\ldots,2^n\right\}$can also be applied on the output layer of each basic network, respectively.

Definition P is defined as the *output transformation gate,* or OTG.

Based on the above analysis, the output conditional probability of entire QPNN in Fig. 5 is

$$Y_k^n = \gamma_i^n y^{(i)n}_{\ \ j} P^{(i)j}_{\ \ k}, \quad i = 1, 2, \ldots, 2^n \tag{14}$$

where i represents the ith basic network, γ_i^n is the probability of the control layer in the state $|i\rangle$, i.e., $\gamma_i^n = p\left(|\gamma^n\rangle = |i\rangle\right)$, $y^{(i)}$ is the output matrix of the ith basic network, and $P^{(i)}$ is the OTG for the ith basic network. Section 6. B will explain why OTGs are significant to improve the classification result.

A unique advantage is that QPNN can be used as memory. Memory is the process by which information is encoded, stored, and retrieved. As mentioned before, in order to encode and store information, the classical data are prepared in a computational basis state by its binary representation. The cost of this strategy is that we need as many qubits as classical bits to represent the data. However, it enables us to store many samples in a superposition state. That is, for n samples $x^1, x^2, \ldots, x^n$, the input layer can be initialized in a superposition state of all samples as follows

$$|x\rangle = \sum_{i=1}^{n} \frac{1}{\sqrt{n}} |x^i\rangle \tag{15}$$

Tracing out all hidden qurons, the reduced density operator ρ of the inputs and outputs is a pure state $\rho = |\phi\rangle\langle\ \phi|$ with

$$|\phi\rangle = \frac{1}{\sqrt{n}} \sum_{i,j} \sqrt{p_i^j}\, |x^i y_j\rangle \tag{16}$$

where p_i^j represents the probability of the output being j for the ith input x^i. Therefore, to retrieved the data, we just need to measure the output layer to obtain a specific label; then, the input layer will collapse to one of the most probable samples of this label by measurement. Moreover, by using amplitude amplify algorithm [28], only $\mathrm{O}\left(\sqrt{n}\right)$ time is required to get the desired label whose probability is $1/n$.

In fact, quantum parallelism can do more than this. Considering the extreme situation, where each qubit in the input layer is initialized in the state $|+\rangle = 1/\sqrt{2}\left(|0\rangle + |1\rangle\right)$, then the input layer state will be a superposition of all possible samples whose amount is far beyond the number of training samples. Then, for each output label, some new input states which are not in the set of training samples can be obtained by measurement.

4 Physical implementation scheme

This section presents quantum circuits of QPNN and a physical implementation scheme to realize these circuits. In particular, we focus on the qubus quantum computer because it can realize the dynamics of qurons efficiently.

Suppose two qubits $|y\rangle$ and $|x\rangle$ are initialized in states $\cos\frac{\theta}{2}|0\rangle + \mathrm{e}^{i\varphi}\sin\frac{\theta}{2}|1\rangle$, where $\theta, \varphi \in \mathbb{R}$ and one of the computation bases $|0\rangle$ or $|1\rangle$, respectively. Apply a control-phase gate $\mathrm{CP}(\omega)$ shown below on them

$$\mathrm{CP}(\omega) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathrm{e}^{i\omega} \end{pmatrix} \tag{17}$$

and followed by an H gate on $|y\rangle$. Then, the joint state becomes

$$\begin{aligned} H_y\mathrm{CP}_\omega\left(|y\rangle\,|x\rangle\right) = &\frac{1}{2}\left(\cos\frac{\theta}{2} + \sin\frac{\theta}{2}\mathrm{e}^{i(\varphi+\omega\cdot x)}\right)|0\rangle\,|x\rangle \\ &+\frac{1}{2}\left(\cos\frac{\theta}{2} - \sin\frac{\theta}{2}\mathrm{e}^{i(\varphi+\omega\cdot x)}\right)|1\rangle\,|x\rangle \end{aligned} \tag{18}$$

where H_y denotes the H gate on qubit $|y\rangle$. Therefore, to realize the quron function Eq. (6), we just need to set $\theta = \pi$ and $\varphi = \frac{\pi}{2} + b$, i.e., $|y\rangle = 1/\sqrt{2}\left(|0\rangle + i\mathrm{e}^{ib}|1\rangle\right)$.

Now for a training sample $x \in \mathbb{R}^m$, if each component of x is represented by d binary bits, then x can be prepared in states of m qurons with d qubits per quron. As shown in Fig. 6a, a series of $\mathrm{CP}\left(\omega_j\right)$ gates are applied between $|y\rangle = 1/\sqrt{2}\left(|0\rangle + i\mathrm{e}^{ib}|1\rangle\right)$ and input qurons $\left|x^j\right\rangle$, $j = 1, 2, \ldots, m$, respectively, followed by an H gate.

Moreover, suppose the component x^j of x satisfies $0 < x^j < 1$ and d binary numbers are used to approximate it, then the detail of a $\mathrm{CP}\left(\omega_j\right)$ gate is shown in Fig. 6b, in which a series of control-phase gates $\mathrm{CP}\left(2^{-k}\omega_j\right)$ are performed on the kth qubit, respectively, $k = 1, \ldots, d$.

Based on the above quantum circuit of qurons, it is easy to construct QPNN of any structure. Figure 7 shows the quantum circuit of the two-layer QPNN in Fig. 3

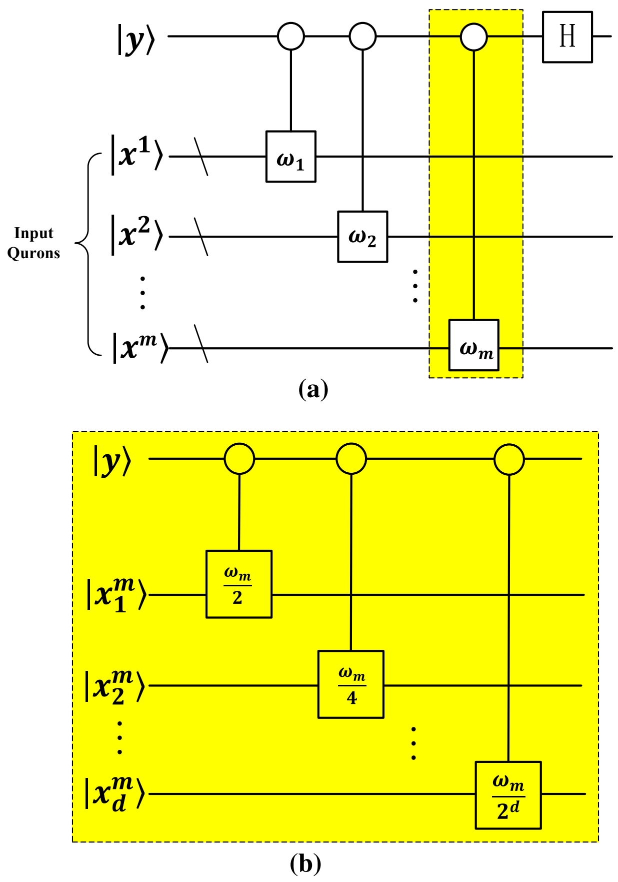


Fig. 6 **a** Quantum circuit to realize a quron. **b** Quantum circuit to realize the CP (ω_m) on quron $\left|x^m\right\rangle$

but with an additional control layer. Each quron in the hidden and output layer is initialized as $1/\sqrt{2}\left(|0\rangle + i\mathrm{e}^{ib}\,|1\rangle\right)$ in which the value b is obtained by the learning algorithm in the next section. The same is true with other parameters of the network. The states of the control layer $|\varphi\rangle$ can be fixed or changed by inputs, which are decided by specific applications. As described in Sect. 3, the control layer is used to generate many different basic networks and linearly combine their outputs. This can be done by quantum multiplexers to perform different $\{\mathrm{cp}\,(\omega)\}$ gates on qurons and different unitary gates $\{P_i\}$ on the output layer. Therefore, suppose $|\varphi\rangle$ contains n qubits, the quantum multiplexers such as Eqs. (19) and (20) are needed.

$$\begin{pmatrix} \mathrm{CP}\left(\omega_i^{(1)}\right) & \\ & \ddots \\ - & \mathrm{CP}\left(\omega_i^{(2^n)}\right) \end{pmatrix} \tag{19}$$

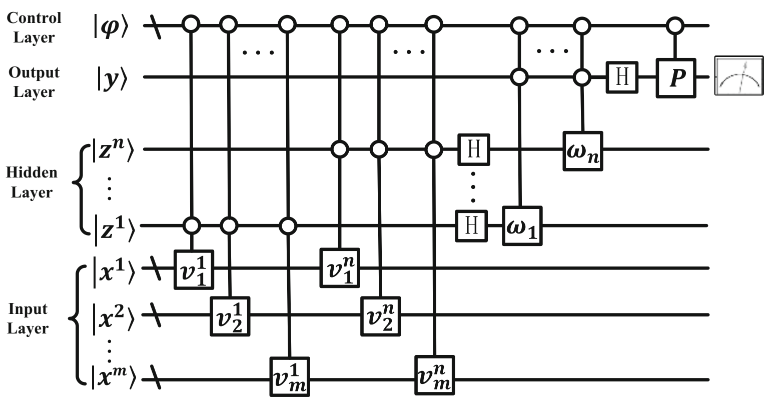


Fig. 7 Quantum circuit to realize a QPNN with a control layer

$$\begin{pmatrix} P^{(1)} & & \\ & \ddots & \\ & & P^{(2^n)} \end{pmatrix} \tag{20}$$

At last, we measure the output layer in the computational basis to obtain the result. Of course, due to the quantum randomness, this process should be repeated several times and choose the most frequent result as the final classification label.

Figure 7 and Eq. (11) show that if the previous layer undergoes some phase errors $R_z(\theta)$, then neither the β_2 nor β_3 will change. Moreover, after interaction with output quron, any bit-flip error X on input qurons will change values of i in β_2 and β_3 simultaneously and thus cannot affect the result of summation. In fact, only the depolarizing channel and amplitude damping [1] will affect the measurement result at any time. Hence, the above analysis implies that QPNN is partial fault tolerant, i.e., robust to some quantum noise.

Now we describe how to realize Fig. 6 using the qubus quantum computer, and this can extend to Fig. 7 easily. A qubus quantum computer is a hybrid system of a processing unit made of qubits and a continuous variable field "bus" that generates interactions and transfers information among qubits. The element operators of interactions in the qubus system can be written in the form [29]:

$$D(\sigma_z\beta) = \exp\left(\sigma_z\beta a^\dagger - \sigma_z\beta^* a\right) \tag{21}$$

where a $(a^\dagger)$ are the field annihilation (creation) operators and $\beta = \chi t e^{i\left(\theta-\frac{\pi}{2}\right)}$; thus,

$$\beta = \begin{cases} -i\chi t, & \text{if}\, \theta = 0 \\ \chi t, & \text{if}\, \theta = \pi/2 \end{cases} \tag{22}$$

where χ is nonlinearity strength and $\theta = O(\pi/2)$ describes coupling of the qubit to the position (momentum) quadrature of the field. According to [25], if we apply the

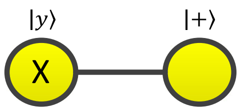


Fig. 8 Measurement pattern to realize the H gate

following sequence displacement operator between qubits 1 and 2, then the sequence performs a geometric phase gate U between them,

$$U\left(\beta_1\beta_2\right) = D\left(\beta_1\sigma_{z1}\right) D\left(-i\beta_2\sigma_{z2}\right) D\left(-\beta_1\sigma_{z1}\right) D\left(i\beta_2\sigma_{z2}\right) = \exp\left(2i\beta_1\beta_2\sigma_{z1}\sigma_{z2}\right) \tag{23}$$

This provides an efficient way to realize qurons because any $U\left(\theta/8\right)$ gate is equivalent to the following operator up to an unimportant constant factor

$$\mathrm{CP}\left(\theta\right)\left[R_{z,1}\left(-\frac{\theta}{2}\right)\otimes R_{z,2}\left(-\frac{\theta}{2}\right)\right] \tag{24}$$

So if we set $\beta_1\beta_2 = \theta/8$, then the additional phase $\exp\left(-\frac{i\theta}{2}\right)$ will be introduced to every pair of qubits in Fig. 6, which can be compensated by setting $b' = b + \theta/2$.

To realize the H gate, the MBQC model [30,31] is used as it can integrate with qubus computer to generate any quantum gates. Firstly, we need to perform a CZ gate between the quron $|y\rangle$ and an auxiliary qubit $|+\rangle$; then, by taking X measurement on $|y\rangle$, qubit $|+\rangle$ will be in state $X^m H\,|y\rangle$, where X^m is a by-product operator due to the randomness of the measurement result m, as shown in Fig. 8.

To construct a CZ gate, we just need to set $\beta_1\beta_2 = \pi/8$. Because $U\left(\pi/8\right)$ is local equivalent to a CZ gate required for cluster-state construction,

$$\mathrm{CZ} \cong U\left(\frac{\pi}{8}\right)\left[R_{z,1}\left(\frac{\pi}{2}\right)\otimes R_{z,2}\left(\frac{\pi}{2}\right)\right] \tag{25}$$

The above equation shows that when qubus is used to generate a cluster state, the additional gate $\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$ is also generated. Thus, to compensate for this additional gate, we just need to initialize $|y\rangle$ in the state $1/\sqrt{2}\left(|0\rangle - \mathrm{e}^{ib}\,|1\rangle\right)$.

In MBQC, we should correct by-product operators X^{m_1} of each gate step by step. However, it is more convenient to let them pass through CP gates as shown in Fig. 9. This is because in QPNNs, any X gates on input qurons after $\mathrm{CP}\left(\theta\right)$ gate cannot affect the result, as described in Sect. 3. Therefore, instead of the $\mathrm{CP}\left(\theta\right)$ gate, $\mathrm{R}_{\mathrm{X},2}CP\left(\theta\right)R_{x,2}$ is performed.

$$\mathrm{R}_{\mathrm{X},2}CP\left(\theta\right)R_{x,2} = U\left(-\frac{\theta}{8}\right)\left[R_{z,1}\left(-\frac{\theta}{2}\right)\otimes R_{z,2}\left(\frac{\theta}{2}\right)\right] \tag{26}$$

Comparing to Eq. (24), the above equation means that to compensate the by-product X gate, we just need to set $\beta_1\beta_2 = -\theta/8$.

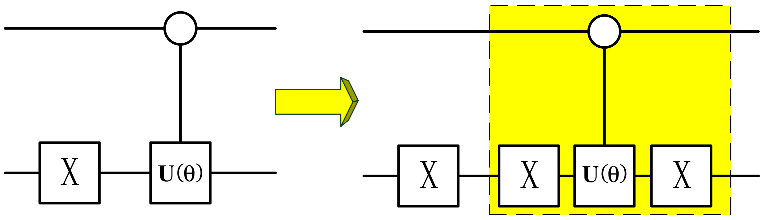


Fig. 9 Diagram to compensate the additional X gate

Fortunately, the method in [32] shows that any block diagonal quantum gates like Eqs. (19) and (20) can be decomposed to the combination of CP gates, H gates, and R_z gates. Similar to the H gate, any R_z gate can be realized by certain measurements on cluster states of three qubits and the by-product X gates are compensated in the same way as before. Thus, all of them can be efficiently implemented on the qubus system.

In summary, all operators needed to perform QPNN are the interaction time adjustment between qubits and the bus. Moreover, as all quantum transformations in each layer can be performed simultaneously, they can be implemented very efficiently if there are enough bus resources on a qubus quantum computer.

5 Learning algorithm

The purpose of the learning algorithm is to find an appropriate set of parameters (weights) that maximize probabilities of the desired results. It is achieved by choosing a loss function $L^n(\omega)$ of the QPNN in advance and then using gradient descent method to minimize it.

Gradient descent is a classical iterative algorithm. We start with an initial value of ω (say, $\omega^{(1)} = 0$). Then, at each iteration, we take a step in the direction of the negative of the gradient at the current point. Eventually, after t iterations, the algorithm outputs the value of $\omega^{(t)}$. To apply gradient descent on a dataset, in practice it is usually better to randomly permute the data and sample without replacement and then repeat. A single such pass over the entire data set is called an epoch.

Therefore, in each epoch of the learning phase, the most complicated operation is the calculation of the gradient of the loss function with respect to the parameters ω of the network. Here we present a back-propagation algorithm that can efficiently calculate gradients. For convenience, we use y_k^n to denote the probability of output layer in $|k\rangle$, i.e., $p(|y^n\rangle = |k\rangle)$.

Firstly, we calculate the output layer weights ω. Rewrite Eq. (11) as follows

$$y_k^n = (\beta_2)_i^n (\beta_3)_k^i \tag{27}$$

In general, for a layer which contains p qurons, $\beta_k^i = \prod_{j=1}^{p} f_{\mathrm{k}_j}\left(i_B \cdot \omega^j\right)$satisfying $k = \left(k_p \cdots k_1\right)_2$. Therefore, the partial derivative of β_k^i with respect to the preactivation value of quron j, i.e., $i_B \cdot \omega^j$, is

$$\frac{\partial \beta_k^i}{\partial (i_B \cdot \omega^j)} = \frac{(-1)^{k_j+1}}{2} \cos(i_B \cdot \omega^j) \Pi_{j' \neq j} f_{k_{j'}}(i_B . \omega^{j'}) \tag{28}$$

Then, if L^n is the loss function of x^n, the gradient of the weight of the quron j at the output layer (layer 3) is

$$\nabla_{\omega^j} L^n = \sum_k \frac{\partial L^n}{\partial y_k^n} \frac{\partial y_k^n}{\partial \omega^j} = \sum_k e_k^n \left(\beta_3'\right)_{k,j}^i (\beta_2)_i^n \, i_B \tag{29}$$

where $e_k^n = \frac{\partial L^n}{\partial y_k^n}$ and $\left(\beta_3'\right)_{k,j}^i = \frac{\partial (\beta_3)_k^i}{\partial (i_B \cdot \omega^j)}$ which can be computed by Eq. (28).

Next, the gradient of the quron j at the hidden layer (layer 2) is

$$\nabla_{\upsilon^j} L^n = \sum_{k,i} \frac{\partial L^n}{\partial y_k^n} \frac{\partial y_k^n}{\partial (\beta_2)_i^n} \frac{\partial (\beta_2)_i^n}{\partial \upsilon^j} = \sum_k e_k^n (\beta_3)_k^i \left(\beta_2'\right)_{i,j}^n x^n \tag{30}$$

It can be shown that the error e_k^n of output layer in the state $|k\rangle$ is back-propagated to the previous layer state $|i\rangle$ by times $(\beta_3)_k^i$, because this quantity is *layer probability transition* from the previous layer to current layer. Moreover, we set $i_B^{(1)} = x^n$ and $\hat{\beta}_1 = 1$. Then, in general, the gradient of the weight of the jth quron at layer $m \quad (m = 2, 3, \ldots)$ is

$$\nabla_{\omega_{(m)}^j} L^n = \sum_k \mathrm{e}^{(m)}{}_k^n \left(\beta_m'\right)_{k,j}^i \left(\hat{\beta}_{m-1}\right)_i^n i_B^{(m-1)} \tag{31}$$

where $\mathrm{e}^{(m)}{}_k^n$ represents the error of layer m back-propagated by the output error e_k^n and

$$\hat{\beta}_{m-1} = \prod_{m'=1}^{m-1} \beta_{m'} \tag{32}$$

represents the LTM from the input layer to layer $(m-1)$.

From Eq. (31), we can see that the most expensive part of learning phase is the summation over all values of i which are the exponential of the number of the hidden layer qurons. Therefore, to accelerate the learning speed, the key is to reduce the effective dimension of layer states. As shown in Eq. (11), if $(\beta_2)_i^n$ is very small for some i, it contributes so little to the result that could be ignored. Thus, in order to focus on states with relatively large probabilities, classical sampling methods can be used to sample the layer. In next section, during the learning phases of two numerical experiments, this strategy is used to trade-off between the learning speed and the accuracy, where for a hidden layer with m qurons, only 2^{m-3} sampling states are needed to calculate.

Alternatively, in a real quantum computer (suppose we have a real quantum computer), this can be done by repeatedly measuring the hidden layer several times to obtain a set of layer states during the learning phase. In fact, the classical methods sample the layer efficiently only in the layer 2 because whose layer state is just a tensor state of its qurons which can be factorized. As a result, each quron can be sampled according to its activation probability ρ, but for a deeper layer, this strategy does not work and only quantum computers perform efficiently.

6 Experiment analysis: Iris data classification and MNIST handwritten digit recognition

In machine learning, the true error of a model can be decomposed into two components: the training (approximation) error and the testing (estimation) error. A good model often means a good trade-off between them. In this section, we use MATLAB to simulate QPNNs for two representative real-life classification applications in feedforward neural network research, i.e., Iris data classification and MNIST handwritten digit database recognition, to verify QPNN's ability of training and testing, respectively.

6.1 Iris data classification

Fisher's Iris data set is one of the best-known sets in the pattern recognition literature. The data set contains three classes of 50 instances each, where each class refers to a type of Iris plant. Four features are measured from each sample: sepal length, sepal width, petal length, and petal width. As shown in Fig. 10, one class in red is linearly separable from the other 2; the latter are NOT linearly separable from each other.

The purpose of the Iris experiment here is to verify the model complexity of QPNN by comparing its ability of nonlinear function fittings to classical FFNN. Classical neural networks are good at fitting functions. In fact, there is a proof that a simple neural network can fit any practical function. Here, all 150 samples are used to train QPNN and classical FFNN and their convergence rates of the training error are compared. The structure of the proposed QPNN is 4-7-2, where four qurons in the input layer represent four features of each sample and two qurons in the output layer are used to be measured with respect to the $\{|0\rangle, |1\rangle\}$ basis to classify three species of Iris. For comparison, a classical FFNN with the structure 4-300-3 is used. The loss function here is the square error $L^n = 1/2\,|\tilde{y}^n - y^n|^2$ where $\tilde{y}^n$ is the training output and y^n is the correct output for the nth sample. By using the back-propagation learning algorithm of Sect. 5, Fig. 4 shows that the QPNN can reach zero training error rate within 500 learning epochs while the classical FFNN can't achieve that even within 1000 epochs. In fact, it reaches zero error around 1700 epochs.

Noting that the essential of QPNN is to approximate desire functions by a series of sine functions, the training error in the learning phase may fluctuate due to the periodicity of trigonometric. However, during the training phase, with the decrease in the derivatives of training error functions, the result tends to stable eventually, which is shown in Fig. 11.

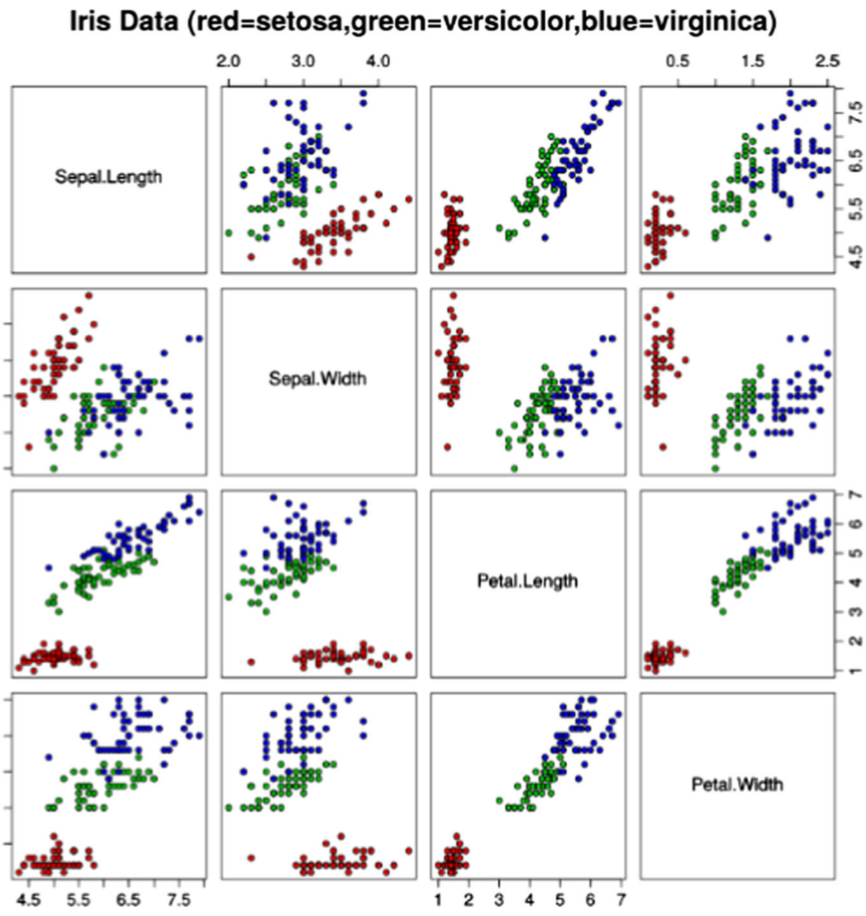


Fig. 10 Scatterplot the Iris data [22]

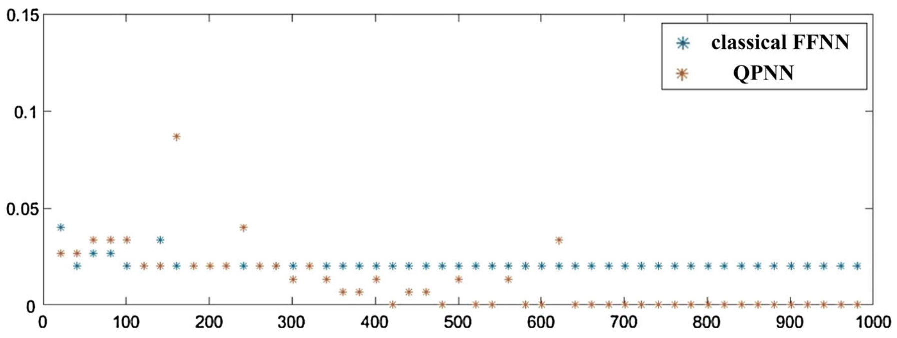


Fig. 11 Training error rates of QPNN and classical FFNN within 1000 epochs

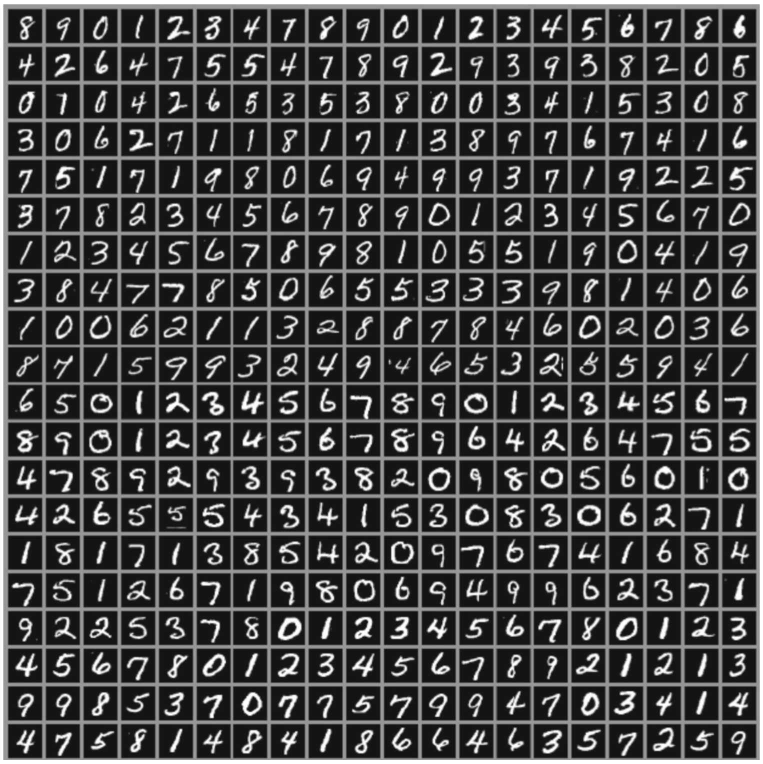


Fig. 12 Examples of the MNIST dataset

6.2 MNIST handwritten digit database recognition

The MNIST dataset consists of scans of handwritten digits and associated labels describing which digit 0–9 is contained in each image, as shown in Fig. 12. This simple classification problem is one of the simplest and most widely used tests in machine learning research which allows researchers to study their algorithms in controlled laboratory conditions, much as biologists often study fruit flies [33]. The MNIST database has a training set of 60,000 samples and a testing set of 10,000 samples. The training sample has been size-normalized and centered in an 28×28image.

The proposed QPNN for MNIST digit recognition is shown in Fig. 13. The structure is as follows:

1. Basic network: There are 784 qurons in the input layer, where each quron is comprised of ten qubits. Hence, the numeric precision of each sample is 2^{-10}. There are ten qurons in the hidden layer and four qurons in the output layer. The reason is that to classify ten classes, four qurons are needed to obtain $2^4 = 16$ classification results.
2. Control layer: Five qurons are used as five control qubits to perform quantum multiplexing, so as to generate $2^5 = 32$ basic networks.

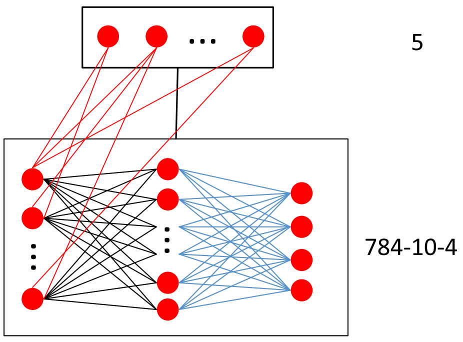


Fig. 13 Proposed QPNN structure

3. Output layer: The OTGs are a set of 16×16 permutation matrices $P^{(i)}$ for $i = 1, \ldots, 32$, which are randomly generated in advance.

Table 1 shows the entire learning phase in which W_j and V are the vectors comprised of all qurons' weights for the jth basic network and control layer, respectively, $j = 1, \ldots, 32$. Therefore, using Eq. (31) in Sect. 5, all gradients of them can be computed. As shown in Fig. 14, a far better result than a QPNN without a control layer can be obtained. For example, a two-layer 784-10-4 QPNN network can only achieve 10.08% ($\sim$0.1) test error. However, using five qurons to generate 32 basic 784-10-4 QPNNs, the experiment shows that we can achieve 2.38% ($\sim$0.02) test error eventually. Moreover, probabilistic outputs can also be useful for the further classification processing: If the measurement results with two highest probabilities are chosen as the candidate labels of test samples, then the test accuracy can be improved to 99.3% in this stage. Of course, the research of the next classifier is beyond the scope of this paper. The experimental result is shown in Table 2, and several published fully connected FFNN results from [23] are also listed for comparison. It can be seen that the proposed QPNN can achieve the best test error rate among those classical ANNs without any additional methods or data preprocessing such as elastic distortion [34]. In QPNN, no additional methods are used, but utilize quantum parallelism to generate many basic networks. However, this strategy does not work well on classical FFNNs because the same structure FFNNs can produce almost the same classification results if they are properly trained under the same input.

Why this strategy works in QPNN? The main reason is that we use permutation matrix p to randomly choose the labels of training samples for each basic network. Generally speaking, the aim of classification is to find the distinction of samples. In classical FFNN, the specific classification label such as "1" or "4" has no real significance. For example, suppose the label of training samples "A" is 1 and of training samples "B" is 3. If we assign the label '4' to "A," then the classification accuracy will not change if the weights of neuron 1 and neuron 4 are interchanged. However, in QPNN, sample "A" and sample "B" differ in one quron since the binary form $1 = (0001)_2$ and $3 = (0011)_2$ differs at the third output quron. But if we assign the label '$4 = (0100)_2$' to "A," then they differ from "B" in three qurons. As a result, weights of QPNN have changed significantly. Therefore, by randomly choosing 10 numbers in $[1, 2, \ldots, 16]$ as labels of training samples, it is possible to train some basic

Table 1 Learning phase of the QPNN

```
Learning Phase
Parameters:
    Number of epochs $\tau$, Step size $\eta$
    The number of weights for the basic network $N_b$ and the control layer $N_c$
    32 permutation matrices $p^{(1)}, \ldots, p^{(32)}$
Input:
    Training samples $(x^m, y^m)$ for $m = 1, \ldots, 60{,}000$
Loss function:
    $L^n = \frac{1}{2}\left|\widetilde{y^n} - y^n\right|^2$
Initialize:
    Choose $W_j^{(1)} \in \mathbb{R}^{N_b}$ at random as the initial weight for the $j$th basic network, $j = 1,2 \ldots, 32$
    Choose $V^{(1)} \in \mathbb{R}^{N_c}$ at random as the initial weight for the control layer.
    32 sets of training samples $S^1, \ldots, S^{32}$, where $S^i = \left\{\left(x^j, y^j \left(p^{(i)}\right)^T\right), j = 1, \ldots, 60{,}000\right\}$
For $i = 1,2,\ldots,\tau$
    For $j = 1,2,\ldots,32$
        Calculate the gradient $g_j^{(i)} = \nabla_{W_j^{(i)}} L^n$
        Update $W_j^{(i+1)} = W_j^{(i)} - \eta \cdot g_j^{(i)}$
    End
    Calculate the gradient $h_j^{(i)} = \nabla_{V^{(i)}} L^n$
    Update $V^{(i+1)} = V^{(i)} - \eta \cdot h_j^{(i)}$
End
Output:
    Weights $W_1^{(\tau+1)}, \ldots, W_{32}^{(\tau+1)}, V^{(\tau+1)}$
```

networks with different weights. In general, different weights mean different features among training samples; thus, the linear combination of them gives a better result than a single network and this explains Fig. 14 to some extent. Thus, in the learning phase, as shown in Table 1, one of the 32 randomly generated permutation matrices $p^{(1)}, \ldots, p^{(32)}$ is used to permute class labels in each of the 32 basic networks, i.e., for the ith basic network, the label vector of the x^j becomes $y^j \left(p^{(i)}\right)^T$. However, as the output conditional probability of the entire QPNN is

$$Y_k^n = \gamma_i^n y_j^{(i)^n} p_{k'}^{(i)^j} \quad i = 1, 2, \ldots, 32 \tag{33}$$

where γ_i^n is the probability of the control layer state $p\left(|\gamma^n\rangle = |i\rangle\right)$ and $y^{(i)}$ is the output probability matrix of the ith basic network. All permuted labels of each basic network will be permuted again to the correct order eventually.

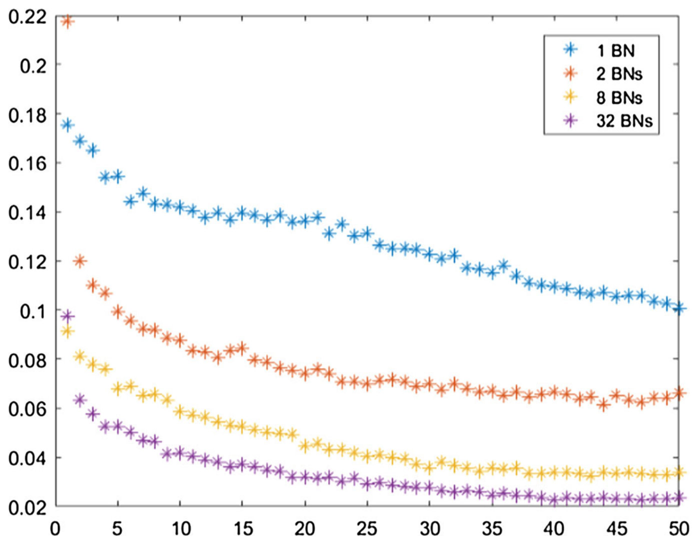


Fig. 14 Test error rates of one BN (basic network), two BNs, eight BNs and 32 BNs within 50 epochs

Table 2 Experimental results of the MNIST

Classify	Test error rate (%)
Two-layer FFNN, 1000 hidden neurons	4.5
Three-layer FFNN, 500 + 150 hidden neurons	2.95
Three-layer FFNN, 500 + 150 hidden neurons [distortions]	2.45
FFNN, 784-500-500-2000-30 + nearest neighbor, RBM + NCA training [no distortions]	1.0
Six-layer FFNN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	0.35
Proposed two-layer QPNN, ten hidden qurons, five select qurons	2.38

The experimental result shows that a resource advantage of QPNN is that only ten hidden qurons and five control qurons are needed to achieve the classical FFNN's accuracy. However, as described in Sect. 4, QPNN can also be used as a probability memory. For example, in the MNIST experiment, if all n training samples are stored in a superposition state $|\psi\rangle = \sum_{i=1}^{n} \left|x^i\right.$, then the probability to obtain the ith sample given the output measurement result j is p_i^j/n as shown in Eq. (16). Therefore, by calculating retrieval probabilities of all samples. Table 3 lists the successful retrieval probabilities of ten labels in which the mean probability is 44.7%. In other words, given a specific label j, it takes average 2.2 times to retrieve a sample which belongs to this label. Moreover, if the input superposition state undergoes quantum noise such as bit-flip channel, this memory can be used as a random sample generator to generate new potential samples which belong to the given labels. In other words, certain quantum noise may be useful in this application.

Table 3 Successful retrieval probabilities of the MNIST experiment

Label	Retrieval probability (%)
0	49.9
1	53.0
2	41.7
3	43.9
4	46.2
5	40.3
6	49.0
7	44.6
8	38.1
9	40.4

7 Conclusion and challenges

Quantum computation and quantum information have enabled us to *think physically about computation*, and this approach has yielded many new and exciting capabilities for information processing [1]. Hence, it is possible to enable us to *think physically about machine learning* , especially *about neural networks*. To this end, we have proposed a novel and partial fault-tolerant quantum neural network model QPNN which can be implemented by quantum computing devices, such as the qubus quantum computer. Furthermore, by preliminary experimental results of MNIST handwritten digit recognition, we have shown that in QPNN, much less neuron resources are needed to obtain a good result compared with classic neural networks. Moreover, the quantum superposition property can make QPNN retrieve the data with a given label and even generate new data.

In fact, the quantum parallelism brings us more insight into machine learning processes. Instead of adding the depth of neural networks, training many networks simultaneously in quantum fashion may be more suitable for today's multi-core parallel hardware before the advent of practical quantum computers. As this paper is the first attempt to design and apply the QPNN for a real-life application, there are some fundamental challenges to solve. Probably the biggest challenge is to establish a genuine quantum learning algorithm for QPNNs, since the best algorithm to deal with all possible layer states should be based on some quantum effects. In addition to this, other challenges are more practical and interesting as well. The first one is the loss function. In this paper, we use square error as the loss function for each sample. However, this function is just better than the cross-entropy function. Indeed, for quantum probabilistic outputs, the type of function indicating the truly classification quality by minimizing its value is still in a question. The second challenge is the method of reducing the dimension of hidden layer state in the learning phase. As described in Sect. 7, classical sampling methods can achieve this purpose, but this strategy is easy to fall into the local optima after a longtime learning. A potential helpful method to solve this problem is the dropout [35], in which some neurons will be dropped randomly during the learning phase. It should be noted that in this paper the value of θ

in Eq. (18) is not obtained by learning as we simply set $\theta = \pi/2$. In fact, θ specifies the sensitive of a quron with respect to input qurons. For example, $\theta = 0$ means the quron is closed or dropped, since the probabilities of active and rest are equal whatever input qurons are. So, developing a quantum dropout method in the learning algorithm will be useful to improve the classification result as well as to reduce the computing cost. Finally, the current additive method used in QPNN to combine basic networks can be further improved. In classical, the adaBoost algorithm [5] works very well for classification by the addition of results of several weak learners. However, adaBoost cannot directly apply to QPNN because its coefficients of addition do not satisfy the normalization condition. So adaBoost is needed to develop its quantum version which use a QPNN as a weak learner. By above improvement in learning algorithm, we can explorer quantum neural networks of more qurons, more layers, more structures and to combine with other machine learning algorithms, such as principle component analysis and convolutional networks.

References

1. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
2. Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
3. Childs, A.M., Landahl, A.J., Parrilo, P.A.: Quantum algorithms for the ordered search problem via semidefinite programming. Phys. Rev. A **75**, 032335 (2007)
4. Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., Preda, D.: A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. Science **292**, 472 (2001)
5. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge (2012)
6. Cristina Diamantini, M., Trugenberger, C.A.: High-capacity quantum associative memories (2015). arXiv:1506.01231v1
7. Schuld, M., Sinayskiy, I., Petruccione, F.: An introduction to quantum machine learning. Contemp. Phys. **56**(2), 172–185 (2015)
8. Wiebe, N., Kapoor, A., Svore, K.: Quantum deep learning. arXiv preprint arXiv:1412.3489 (2014)
9. Wiebe, N., Braun, D., Lloyd, S.: Quantum algorithm for data fitting. Phys. Rev. Lett. **109**, 050505 (2012)
10. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum algorithms for supervised and unsupervised machine learning (2013). arXiv:1307.0411
11. Menneer, T., Narayanan, A.: Quantum artificial neural networks vs classical artificial neural networks: experiments in simulation. In: Proceedings of the IEEE Fourth International Conference on Computational Intelligence and Neuroscience, pp. 757–759 (2000)
12. Ventura, D., Martinez, T.: Quantum associative memory. Inf. Sci. **124**(1–4), 273–296 (2000)
13. Pagiamtis, K., Sheikholeslami, A.: Content-addressable memory (CAM) circuits and architectures: a tutorial and survey. IEEE J. Solid State Circuits **41**(3), 712–727 (2006)
14. Behrman, E.C., Nash, L.R., Steck, J.E., Chandrashekar, A.A., Skinner, S.R.: Simulations of quantum neural networks. Inf. Sci. **128**(3), 257–269 (2000)
15. Panella, M., Martinelli, G.: Neural networks with quantum architecture and quantum learning. Int. J. Circuit Theory Appl. **39**, 61–77 (2011)
16. Schuld, M., Sinayskiy, I., Petruccione, F.: The quest for A quantum neural network. Quantum Inf. Process. **13**(11), 2567–2586 (2014)
17. Sahni, V., Patvardhan, C.: Iris data classification using quantum neural networks. In: AIP Conference Proceedings, vol. 864, p. 219 (2006)
18. Fei, L., Baoyu Z.: A study of quantum neural networks. In: IEEE International Conference on Neural Networks and Signal Processing, December (2003)
19. Weinberg, S.: Precision tests of quantum mechanics. Phys. Rev. Lett. **62**, 485–488 (1989)

20. Silva, A.J.D., Ludermir, T.B., Oliveira, W.R.D.: Quantum perceptron over a field and neural network architecture selection in a quantum computer. Neural Netw. **76**, 55–64 (2016)
21. Schuld, M., Sinayskiy, I., Petruccione, F.: Simulating a perceptron on a quantum computer (2014). ArXiv:1412.3635
22. Iris flower data set. https://en.wikipedia.org/wiki/Iris_flower_data_set
23. THE MNIST DATABASE of handwritten digits website. http://yann.lecun.com/exdb/mnist/
24. Chen, J., Wang, L., Charbon, E., Wang, B.: A programmable architecture for quantum computing. Phys. Rev. A **88**, 022311 (2013)
25. Brown, K.L., Thesis, P.D.: Using the Qubus for Quantum Computing. University of Leeds, Leeds (2011)
26. https://en.wikipedia.org/wiki/Einstein_notation
27. Jesse, A.: Garman: A Heuristic Review of Quantum Neural Networks, master paper of Imperial College London (2011)
28. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation (2000). arxiv:quantum-ph/0005055
29. Spiller, T.P., Nemoto, K., Braunstein, S.L., Munro, W.J., van Loock, P., Milburn, G.J.: Quantum computation by communication. New J. Phys. **8**, 30 (2006)
30. Raussendorf, R., Briegel, H.J.: A one-way quantum computer. Phys. Rev. Lett. **86**, 5188 (2001)
31. Raussendorf, R., Browne, D.E., Briegel, H.J.: Measurement-based quantum computation with cluster states. Phys. Rev. A **68**, 022312 (2003)
32. Shende, V.V., Bullock, S.S., Markov, I.L.: Synthesis of quantum logic circuits. In: IEEE Transaction on CAD, vol. 25, no. 6 (2006)
33. Shwartz, S.S., David, S.B.: Understanding Machine Learning. Cambridge University Press, Cambridge (2014)
34. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. **29**, 141 (2012)
35. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. http://jmlr.org/papers/v15/srivastava14a.html, http://www.cs.toronto.edu/~hinton/absps/dropout.pdf